Übungsblatt 3

Ralph Gauges Ursula Rost Katja Wegner

07.11.2007

Aufgabe 1:

Nehmen Sie die Funktionen die Sie für die Aufgaben vom 31.10. geschrieben haben und teilen Sie diese auf in eine Header Datei mit den Deklarationen und eine Source Datei mit der Implementierung. Die Funktionen lassen sich dann einfach in weiteren Projekten verwenden.

Aufgabe 2:

Schreiben Sie eine Klasse Bruch, welche eine rationale Zahl (Bruchzahl) abbildet. Der Zähler und der Nenner sollen vom Typ **unsigned int** sein, das Vorzeichen des Bruchs soll in einem Attribut vom Typ **bool** gespeichert werden. Die Klasse soll Methoden haben um zwei Bruchzahlen zu addieren, zu subtrahieren, zu multiplizieren und zu dividieren. Die Funktionen sollen jeweils zwei Argumente vom Typ *Bruch* entgegennehmen und das Ergebnis als Wert vom Type *Bruch* zurückliefern. Die jeweiligen Ergebnisse müssen keine gekürzten Brüche darstellen. Die Klasse soll in eine Header Datei für die Deklarationen und eine Source Datei für die Definitionen aufgeteilt werden. Gleiches gilt für eventuell zusätzlich benötigte Funktionsdeklarationen und -definitionen. Die *main* Funktion soll vom Benutzer vier separate, maximal zweistellige Zahlen entgegen nehmen, wobei die ersten beiden Zähler und Nenner der ersten Bruchs sind und die letzten beiden Zähler und Nenner des zweiten Bruchs. Damit soll dann jeweils eine Instanz der Klasse Bruch erzeugt werden und das Ergebnis der Addition, Subtraktion, Multiplikation und Division der beiden Brüche ausgegeben werden. (Die Eingabe der Zahl durch den Benutzer soll das Programm auf Korrektheit überprüfen und die

Eingabe in Zahlenwerte umwandeln mit denen die Brüche initialisiert werden. Bei fehlerhafter Eingabe ist eine entsprechende Fehlermeldung auszugeben. Achten Sie insbesondere auch darauf, dass der Benutzer sinvvolle Werte für Zähler und Nenner eingibt.)

Aufgabe 3:

Schreiben Sie eine Klasse Viereck. Die Klasse soll als Attribute vier 2D Koordinaten speichern, d.h. jeweils ein x und ein y Wert vom Typ **float** und ein Attribut, ebenfalls vom Typ **float**, welches den Umfang speichern soll. Die Klasse soll ferner über folgende Methoden verfügen:

init initialisiert alle Koordinaten und en Umfang mit dem Wert $-1.0f$.

berechneUmfang berechnet den Umfang des Vierecks und speichert das Ergebnis in dem entsprechenden Attribut

berechneLaenge nimmt als Argumente 2 Koordinaten (4 **float** Werte) und berechnet die Länge der Strecke zwischen den Koordinaten und gibt das Ergebnis als **float** Wert zurück

getUmfang liefert den Wert des Attributes zurück, falls dieses noch nicht berechnet wurde, soll vorher *berechneUmfang* aufgerufen werden

Standard Konstruktor ruft die init Methode auf

Konstruktor mit 8 float Argumenten setzt die Koordinaten auf die angegebenen Werte und berechnet den Umfang

sonstiges Methoden zum Setzen und Auslesen der 4 Koordinaten

Schreiben Sie nun eine Klasse Rechteck, welche von Viereck abgeleitet ist und drei zusätzlichen Attribute für die Breite, die Höhe und die Fläche hat. Die Attribute sollen ebenfalls in einer Methode namens *init* mit dem Wert $-1.0f$ initialisiert werden. Der Umfang und die Flaeche sollen anschliessend im Konstruktor berehnet werden. Zum Berechnen der Werte kann man die geerbte Methode *berechneLaenge* verwenden. Ausserdem soll die Klasse über eine Methode verfügen, welche die Fläche des Rechtecks berechnet und im entsprechenden Attribut speichert. Die Breite, Höhe und Fläche sollen über

Methoden der Klasse auslesbar sein. Es soll wiederum einen Standard Konstruktor geben, welcher *init* aufruft und einen Konstruktor, der 4 **float** Werte übernimmt und die Position, sowie die Breite und Hoehe setzt.
Von der Klasse Rechteck soll nun noch eine Klasse Quadrat abgeleitet werden, welche die Methoden zur Berechnung der Fläche und des Umfangs, passend für ein Quadrat, neu implementiert. Ausserdem soll es wieder einen Standard Konstruktor geben und einen Konstruktor, welcher die Position und die Breite als 3 **float** Werte übernimmt.
Die Methoden zum berechnen sollen von ausserhalb der Klasse nicht benutzt werden können. Dasselbe gilt natürlich für die Attribute. Zum Lösen dieser Aufgabe benötigen Sie einige mathematische Funktionen, z.B. um die Quadratwurzel aus eine Zahl zu ziehen. Dazu müssen sie die Datei ¡math.h¿ einbinden, dort gibt es die Funktion *sqrt*. Diese nimmt ein Argument vom Typ **float** und liefert als Ergebnis auch einen **float** Wert zurück.

Aufgabe 4:

Stellen Sie Sich vor, Sie werden vor das Problem gestellt, eine Medienverwaltung für eine Bibliothek zu schreiben. Die Bibliothek verfügt über folgende Medientypen: Bücher, Zeitschriften, Musikkasetten, Musik CDs, Musik DVDs, Schallplatten, Video DVDs und Videokasetten. Finden Sie eine geeignete Klassenhierarchie und überlegen Sie Sich welche Attribute und Methoden für die einzelnen Klassen sinnvoll sein könnten. (Tipp: Überlegen Sie welche Gemeinsamkeiten und Unterschiede die einzelnen Medientypen haben. Z.B. Kann man vielleicht alle diese Dinge ausleihen und die Bibliothek muss vermerken, wer was wann ausgeliehen hat, auf der anderen Seite kann man sicherlich nur für Bücher und Zeitschriften eine Seitenzahl sinnvoll speichern.) Versuchen Sie anschliessend diese Klassenhierarchy zu implementieren.

Aufgabe 5:

Die sogenannte Standard Template Library in C++ stellt dem Programmierer eine Reihe von häufig verwendeten Strukturen (Klassen) zur Verfügung, darunter auch eine Klasse für Strings. Da diese sicherer ist und mehr Funktionalität bietet als die C String (**char** Felder) die wir bisher verwendet haben, wollen wir ab sofort diese Klasse anstelle der C Strings verwenden.

Um C++ Strings zu verwenden muss man die Deklaration der Klasse in das Programm einbinden, diese befinden sich in der Header Datei *string*, d.h. um C++ Strings in einem Programm zun verwenden muß man die Deklaration mit `#include <string>` in den Quellcode einbinden. Ein weiterer Vorteil von C++ Strings ist, daß man sich nicht im Vorraus Gedanken über dessen Länge machen muß. Ansonsten ist die Benutzung von C++ Strings der von C Strings sehr ähnlich. Man sollte jedoch beachten, daß C++ Strings im Gegensatz zu C Strings keine 0 am Ende haben. C++ Strings verfügen über viele Methoden, um Strings zu manipulieren und zu untersuchen.
Hier ein kleines Beispiel:

```
1  #include <string>
2  #include <iostream>
3
4  int main()
5  {
6      std::string name;
7      std::cout << "Bitte geben Sie Ihren Namen ein: ";
8      getline(std::cin,name);
9      std::cout << "Sie heissen: " << name << "\n";
10     std::cout << "Ihr Name hat: " << name.size() << " Buchstaben.\n";
11     std::cout << "erster Buchstabe: " << name[0]  << "\n";
12     std::cout << "letzter Buchstabe: " << name[name.size()-1]  << "\n";
13     return 0;
14 }
```

Zeile 1: Bindet die Deklaration für C++ Strings ein

Zeile 6: Deklariert eine Variable vom Type *std::string* mit dem Namen *name*. Die Variable ist somit eine Instanz der Klasse *std::string*. Instanzen werden durch den Standard Konstruktor initialisiert, d.h. nach Zeile 6 enthält der String *name* einen Leerstring.

Zeile 7: Fordert den Anwender dazu auf, seinen Namen einzugeben.

Zeile 8: Liest den Namen vom Anwender mit Hilfe der *getline* Funktion ein. Die Syntax ist aber diesmal eine etwas Andere. Zum ersten ist dieses *getline* keine Methode von *std::cin*, sondern eine Funktion, welche zwei Argumente übernimmt. Das erste Argument ist der sogenannte stream aus dem gelesen wird, in diesem Fall *std::cin*, und das zweite Argument ist das Objekt, in welchem das Ergebnis gespeichert wird. Da der C++

String *name* eine variable Länge hat, braucht man auch die Länge der einzulesenden Zeichen nicht angeben.

Zeile 9: Gibt den Namen aus.

Zeile 10 Gibt die Länge des Strings aus, welche man man mit Hilfe der Methode *size* der Klasse *string* ermittlen kann.

Zeile 11 und 12: Geben jeweils den ersten und letzten Buchstaben des Namens aus. Die einzelnen Buchstaben eines C++ Strings sind wiederum Daten vom Typ **char**, und man kann über den Namen der Stringvariablen mit nachgestellten Index in eckigen Klammern auf die einzelnen Zeichen zugreifen, genau wie bei C Strings. Der Index beginnt wiederum bei 0, d.h. der Index des letzten Zeichens ist `name.size()-1`.

Versuchen Sie die Aufgabe 1 und 2 vom zweiten Übungsblatt so umzuschreiben, daß diese nun C++ Strings verwenden.